Chapter 2

Data Workflows

One of the most common use cases for coding in libraries is data processing. Whether it's import/export, quality control, combining data from different sources, or adapting externally provided records to local purposes, data tasks are ubiquitous in library technical services. Many of them are quite repetitive and, as such, lend themselves well to scripting. In addition, computers are often faster and more accurate than humans at repetitive tasks. Therefore, the time spent in developing data processing scripts can pay off manyfold in increased efficiency, freeing librarians to do more creative, sophisticated tasks that require human insight.

In this chapter, I'll provide an overview of eight scripts that simplify various data processing tasks and do a deep dive into a ninth. Their use cases include metadata quality control, import/export workflows, bulk downloading, and data migration.

It's notable that these nine scripts are in seven different programming languages (bash, Python, VBA for Excel, Perl, Ruby, XSLT, PHP). Beginning programmers often want to know the best language to learn, and there truly isn't one. While some languages may be easier or harder for a given student, and more or less suited for a particular use case, they all incorporate the same fundamental programming concepts, and all of them open a lot of doors.

Examples

Facing a need to export data from DSpace, nina de jesus wrote a bash script to do it. This script exports metadata from every handle in a series and dumps it to a CSV file for later processing. Like many programmers, de jesus learned how bash worked in the course of getting this script to work. This made it slow going at first, but she expects it to pay off handsomely over time: "For all that this tiny script took me a long time to write (maybe three or four days to get it working properly), it saved me a lot of tedious hours of slowly (manually) going through database tables and spreadsheets to get the data I needed. And now I can use the script whenever I need to get this kind of data out of DSpace again (which I'm sure will happen)."

nina de jesus's script
http://satifice.com/2014/10/22/exporting-the-metadata-of-a-range-of-handles-in-dspace

Hillel Arnold also needed to export metadata: in his case, EAD files from ArchivesSpace. His short Python script finds all the resource IDs that match a given criterion, gets their EAD, and writes it to a specified destination.

Hillel Arnold's script
https://gist.github.com/helrond/1ef5b5bd47b47bd52f02

Becky Yoose also saved time by automating a tedious workflow. Her library had a trigger file, in Excel format, of books to be acquired under a patron-driven acquisition policy. The library needed to extract MARC records from the database using local control numbers in the file, edit them for consistency with local cataloging rules, and insert codes to make the ILS's purchasing module automatically create an

order request. By hand, this workflow took five to ten minutes per week per record, or almost one to two hours per week of cataloger time; the script reduced processing to two minutes total, for a net savings of one to two weeks per year of cataloger time. Additionally, as she notes, "Each time a human has to touch the record, it's a possible fail point" because of the risk of misspellings and other oversights; machine processing improves accuracy while saving time.

A version of this script (edited for use in the LITA/ALCTS Library Code Year Interest Group Python preconference at ALA Annual 2013) is available online. The README at that link explicitly permits library reuse and adaptation.

Becky Yoose's script (edited)
https://github.com/LibraryCodeYearIG/MARC-record-edit

Tricia Lampron had a text file with bar codes corresponding to files that needed to be downloaded. By hand, this meant she had to "enter in the link, right click to download the file, and then . . . change the file name once downloaded" for up to 190 files—a tremendously tedious process. Her Python script reads the text file, constructs the corresponding URL, downloads the file, and creates an appropriately named XML file locally.

Joy Nelson and Ruth Szpunar both faced metadata cleanup tasks. Szpunar cleaned up and organized metadata from a digitization project using VBA for Excel. Nelson works for an ILS support vendor whose customers often want to move data from one MARC tag to another during ILS migrations; she wrote a Perl script to handle this task. Nelson's use case in particular underscores how tedious, repetitive tasks can be great scripting candidates if you can specify a clear rule for them; as long as you can specify the exact field and subfield that you want data to move from and to, you can write this program with only a handful of lines of code, and it will execute accurately over thousands of records in (almost) no time.

Misty De Meo faced a more complex migration problem. She inherited a controlled vocabulary, "but it became clear that there were a large number of deficiencies in it: inconsistencies, missing terms, duplicate terms, incorrectly-matched relationships, and so on." It's difficult to have a computer fix this kind of problem because there are so many ways the data can be wrong, and making it right could require human judgment calls. However, she was able to write a Ruby script that automatically fixed the simpler errors and flagged others for subsequent human review. Along the way she gained better insight into her data set: "It really helped me understand why the metadata had problems, and helped me reason about what was probably intention vs what was probably an accident. Many patterns that weren't at all obvious when reading metadata by eye instantly became clear once it was being processed by software."

These kinds of data quality problems are common in library coding and can sometimes be so pervasive or frustrating as to make it infeasible to build software on top of the data. However, exposing problems through attempts to write code can suggest opportunities for improvement. Clarifying local cataloging rules, adding input validation (see Eric Phetteplace's script in chapter 4), or writing scripts that run regularly to detect common problems can all improve data quality, for example.

Annie Glerum also had metadata quality problems: in her case, inaccurate vendor records. Her XSLT stylesheet "identifies records needing location code edits for the catalog's holdings record, corrections to the MARC coding, edits to bring the record to full level, or human review for special formats and sets." It outputs its report as an Excel spreadsheet, which fits well into subsequent workflows.

Deep Dive: LibALERTS

Patrons at Westlake Porter Public Library (Westlake, OH) wanted to be notified by text message when the library got new books by their favorite authors.[1] While the library's OPAC had similar functionality, it didn't let patrons refine their searches enough to be useful and had been turned off. The library's Drupal website, however, provided many of the building blocks needed: SMS integration, a module to create Drupal nodes from MARC imports, and a module allowing users to subscribe to terms in the site taxonomy. Matt Weaver—"a development team of 1 [with] a budget of 0"[2]—was able to build a prototype alerts service by combining these modules.

However, he quickly found that he had to address data quality issues before the service could be offered to patrons. Publisher-provided MARC records did not consistently handle middle initials and sometimes misspelled author names, meaning that a single author could be represented by a variety of terms. All these terms needed to be combined in order to offer patrons a single term they could subscribe to.[3] This single term is an element of his site's taxonomy, which in turn is generated from MARC records in the Drupal site (which are distinct from MARC records in the catalog). Therefore, Weaver needed to compare his publisher-provided MARC records with his catalog versions and create records with canonical names that he could feed into his Drupal site.

In Weaver's marcreupload repository, the `marc_upload_page` script provides a front end for submitting MARC records, including a Levenshtein distance function that automatically suggests several

close-match spelling options for author names. Records submitted through this page are processed by the `authorchange` script, which we examine here. Follow along at this link:

Matt Weaver's `authorchange` script
https://thatandromeda.github.io/ltr/Chapter2.html

Line 1 simply tells the computer that this is a PHP script. **Line 2** includes the PHP library for processing MARC records, which we'll need later.

Lines 3–14 harvest data from the MARC record submitted through the form on `marc_upload_page` and store it as variables for later use. An important variable here is `$closest`; this is an array of the author names from our catalog that are the closest match to the author names submitted in the form.

Lines 15–19 write HTML; this is the web interface presented to the human using the script, and it's how we'll display feedback. (Keep in mind that this script also has a machine audience: the Drupal site that will be consuming the MARC records it generates.) All subsequent lines that begin with `echo` are also writing HTML, which provides feedback to the user, and will be skipped in this read-through.

Line 21 initializes the `$arraypos` variable; this is how we'll keep track of how many times we've iterated through the upcoming loop.

In **line 22**, we begin the loop that will take up the remainder of the program. Broadly speaking, what we'll do in this loop is look through each submitted author name, compare it to the corresponding closest-match name, and create records for the Drupal site if it seems correct to do so.

In **line 23** we increase the `$arraypos` counter by one (the ++ syntax, meaning "increase this number by 1," is common to many languages). In this loop, we're processing several records. When we generate a new MARC record for the first author name, we want to make sure we're comparing it against the first name in the closest-match array and using MARC field data from the first submitted MARC record (and so on for the second and subsequent records). Keeping track of this counter lets us be sure to look in the right place for all our information.

In **line 28**, we check to see if the author name we're currently examining is the same as the corresponding closest-match name. If it is, we'll write a MARC record; if it's not, we'll skip processing—this is a case that requires human judgment.

Assuming the names match, in **line 32**, we create an empty MARC record, which we'll call `$marc`. **Line 33** sets its leaders to be the same as those in the submitted record. **Lines 34–37** create a new MARC 008 field using the same information as in the 008 field of the submitted MARC record. (Note that we use the `$arraypos` variable to select the first, second, etc., from the array of submitted 008 fields, as appropriate.) We then append that field to `$marc`.

Lines 38–60 proceed in the same manner, copying data from the submitted MARC record to the new one being created. **Line 44** varies this slightly, using the author name from the closest-match array (which, you recall from **line 28**, exactly matches the submitted author name).

In **line 63**, we check to see if we've successfully generated a MARC record. If so, we tell the user we've written a file for it; if not, we inform the user accordingly.

Lines 66–69 actually write the MARC record for our Drupal site to our output file.

Line 72 connects back to the `if` condition that we opened in **line 28**. All the lines since then have been handling the case where the author name and the closest-match name are the same. The `else` in this line switches us to the alternative case. If we don't have matching names, we can't generate an authoritative record, so we simply inform the user of this (**line 74**) and move on. The remaining lines close all our unclosed code blocks to complete the program.

Weaver's script underscores several issues of software development process that numerous respondents commented on. One is the importance of looking for existing code rather than building from scratch. Although Weaver did write several scripts in the process of getting LibALERTS to work, the vast majority of the service resides in Drupal modules already written by others; his code patches them together. Many people, with development budgets similar to Weaver's, will find that this is much more achievable than writing things from scratch. It's often better practice, too, since existing modules benefit from the development expertise and user testing of large communities and get quicker and more thorough bug fixes than in-house code produced with limited labor.

Another issue is iteration. Very few programs work right the first time. Even if they're bug-free (which is rare), developers usually can't envision exactly what users might need to do or all the special cases the code might end up needing to address. In this case, Weaver discovered the data quality issues by writing the prototype version of his code and seeing where it encountered problems. The `authorchange` script is part of how he solved those problems.

This shouldn't be viewed as a failure of the first script, by the way. Fred Brooks, in his classic of software project management, *The Mythical Man-Month*, said you should expect at least half your time to be spent on testing and debugging—and the more components you find yourself integrating, the longer the overall time to completion.[4] Planning for iteration is simply responsible software practice.

So how would you iterate this program from here? Things to try include these:

- As a Pythonista with limited PHP skills, I had trouble reading this program and found myself reformatting it in order to analyze it. In particular, I reflexively applied semantic whitespace—indenting the contents of *for loops* and *if conditions* to make the code blocks stand out more clearly on the page. How could you reorganize and comment the `marc_upload_page` script to make it easier to read?
- Determine: is copying the leaders from the existing record valid? If we've changed author names or failed to preserve any MARC fields between `marc_upload_page` and `authorchange`, the leaders no longer accurately represent the length of the file. The `setLeader` function from PHP's MARC library explicitly does not perform any validation, so we can easily end up with invalid leaders. Figuring out if this is a problem in our case requires analyzing `marc_upload_page` and considering the input data (which may vary in different contexts, depending on local cataloging practices). If we can't safely copy the leaders, what should we do instead? (Alternatively, we could skip the entire analysis if we simply planned to generate leaders rather than copy them. Consult PHP's MARC module source code and documentation to see if it has that functionality.)

`Record.php`, containing the `setLeader` function, from PHP's MARC library
https://github.com/pear/File_MARC/blob/master/File/MARC/Record.php

- The `marc_upload_page` and `authorchange` functions handle authors in the 100 field, but don't handle additional authors from the 700 field. However, patrons who are interested in new works by particular authors may want to see their coauthored works as well. How can we add support for this?

Scripts in This Chapter

nina de jesus's script
http://satifice.com/2014/10/22/exporting-the-metadata-of-a-range-of-handles-in-dspace

Hillel Arnold's script
https://gist.github.com/helrond/1ef5b5bd47b47bd52f02

Becky Yoose's script (edited)
https://github.com/LibraryCodeYearIG/MARC-record-edit

Matt Weaver's `authorchange` script
https://thatandromeda.github.io/ltr/Chapter2.html

Notes

1. See Westlake Porter Public Library, LibALERTS webpage, accessed December 15, 2014, www.westlakelibrary.org/libalerts.
2. Matt Weaver, "LibALERTS: An Author-Level Subscription System," *Code4Lib Journal*, no. 18 (October 3, 2012), http://journal.code4lib.org/articles/7363.
3. See Weaver's *Code4Lib* article (cited in note 2) for sample code handling this issue.
4. Frederick P. Brooks Jr., "The Mythical Man-Month," in *The Mythical Man-Month: Essays on Software Engineering, Anniversary Edition*, 20, (Boston: Addison-Wesley, 1995).